CS5300 Project 2

1. **Important Issues for graders**

1.1 Location and format of pre-filtered file

Location:
https://s3.amazonaws.com/edu-cornell-cs-cs5300s16-ys684-project2/elasticmapreduce/input/inputFile.txt
Format: sourceNodeID initialPageRank
destinationNodeID1,destinationNodeID2,destiantionNodeID3,…

1.2 source files

Source files of Simple PageRank, Jacobi Blocked PageRank, Gauss Blocked PageRank and Random Blocked PageRank are included in separated folders under solution.

For Simple PageRank, main function is in SimplePageRank.java, for Jacobi Blocked PageRank, main function is in BlockedPageRank.java. For Gauss Blocked PageRank, main function is in GaussBlockedPageRank.java. For Random Blocked PageRank, main function is in RandomBlockedPageRank.java.

The program which filters the edges.txt is stored inside the preFilterProgram folder.

1.3 extra points

We implemented both Gauss-Seidel Blocked PageRank and also Random Blocked PageRank, the comparison of all the results can be found in section 8 of our README. Also in the result folder under solution.

2. **Overall Structure**

The whole project consists of data filtering, simple PageRank application, blocked PageRank application and test in EMR.

3. **Data Filtering**

3.1 Parameters:

netID = dh626
rejectMin = 0.5634
rejectLimit = 0.5734
Number of selected edges = 7525098

3.2 Operations

First, download “edges.txt” from AWS S3 to local.
Then set the inputPath of “edges.txt” and the outputPath of the computed “inputFile.txt”.
(By setting the static fields in class nodefilter:
Static String inputPath = “/{$ your local path to edges.txt}/egdes.txt”;
Static String outputPath = “/{$ your desired local path for output}/inputFile.txt”;)
Then run the nodefilter.java. The selected and computed “inputFile.txt” file will be located in the path indicated by outputPath.

3.3 Output Format

For each node in the graph, the output format is as follows:
sourceID initialPageRank destinationID1,destinationID2,destiantionID3,…
The initialPageRank is computed by using 1 divided by the total number of nodes in the graph (685230).
If for any node, none of its edge is selected, it will still be included in the output file, but with a empty destination list.

3.4 Logic

Read in the “edges.txt” line by line. Use a method selectInputLine(doubel x) to judge if the current line being read falls between rejectMin and rejectLimit. If selected, put this line’s info into a HashMap keyed by the source node ID. The value of HashMap is the list of IDs of destination nodes for corresponding source node. Then output the info kept in HashMap in the format described in 3.3.

4. Simple PageRank

4.1 Mapper

Functionality:Receive Node information and send edge information to destination Node.

InputKey: Text. A line in the input file, which contains information of NodeID, current PageRank value and a list of NodeIDs represent the destination of outbounding edges.

InputValue: Text

OutputKey: LongWritable(NodeID)

OutputValue: Text. Text is used as the type of mapper output value. Except for sending the PageRank value to the destination node, mapper will also send the information of the whole node, which is the same as the input value, to the reducer for the reconstruction of the graph in future MapReduce passes.

4.2 Reducer

Functionality: Receive pagerank values of all incoming edges for each node, calculate new PageRank value for this node and update the global residual. After such calculation, node information with updated PageRank value and list of outbounding edges will be stored into output txt file.

InputKey:LongWritable (NodeID)

InputValue: Iterable<Text>. There are two kinds of input value with different length and different information, one is the PageRank value of edge coming to this node. The other one is the previous PageRank value and list of outbounding edges of this Node.

OutputKey: Text.

OutputValue: Text, which contains NodeID, updated PageRank value and a list of outgoing edges.

4.3 Residual Calculation

Counter is used in simple PageRank to calculate the residual. Each time after a Node has updated its PageRank value, the current residual of this node will be calculated and add to the global counter by calling **context.getCounter().increment()** with specified arguments. Since the type of counter is long and the required precision of residual is 4 decimal digits, we first multiply the residual by 10,000 and then add it to counter. When all the reduce tasks has finished, the counter is divided by 10,000 and 685230 and then compared to 0.001, which is the limit of average residual error. We decide whether to continue or stop mapreduce passes by the comparison above.

5. Jacobi Blocked PageRank

5.1 Mapper

Functionality: Mapper deal with processed Node Information from the input file, containing NodeID, PageRank value and a list of destination NodeIDs. A method in class BlockMatch.java will be called to get the block ID for each node. The core function of mapper is to send edge information to the block which contains the destination node. Besides, node information will also be sent to reducer for graph reconstruction for inner iterations in reducer. A detail that needs to be mentioned is that when sending edge information, it is necessary to differentiate whether the edge is coming from in-block node or out-block node, since the PageRank value of edges from outside the block will not change for inner iterations in reducers.

InputKey: Text. Containing a line of data in input file.

InputValue: Text. Empty.

OutputKey: LongWritable. This is the block id. So that the reducer can receive all the information of nodes and incoming edges within this block.

OutputValue: Text. Contains information of both nodes and incoming edges inside or to this block.

Get block id by node id: this functionality is realized by class blockSearch. This class includes a long array as well as a method blockIDofNode(long n). Each element in the array represents a block. The block ID is the array element index plus 1. The value in the element is the last nodeID in this block. Each time when a nodeID is taken into method blockIDofNode(long n), a binary search will be implemented by comparing nodeID with the values in array elements, which defines nodeID range between blocks. Finally the index of array element, in whose range this nodeID lies, will be returned. Because this function is based on binary search, it can run in O(log(numberOfBlocks)) as required in instruction.

5.2 Reducer

InputKey:LongWritable. This contains the block id.

InputValue:Iterable<Text>. By iterating the inputValue, both the node information of the block and the edges with destination belong to this block has been stored for inner iterations.

OutputKey:Text, which is null in our program.

OutputValue: Text: The node information with updated PageRank value and a list of outgoing edges.

Store input data: In reducer, all the input values are iterated. Node’s outgoing edges and initial PageRank values are stored into corresponding hashmaps. Besides, the incoming edges from nodes inside the block and from nodes outside the block are stored separately in two different hashMaps.

Inner Iteration Control: In inner iterations, we first update the PageRank value of each node in current block and store them in the nextPR HashMap. Then the edges from inside block nodes have been updated according to the newly PageRank value of each node inside the block. Then inside block average residual is calculated and compared to 0.001 to decide whether to continue or stop. When the inner iteration stops, reducer will update the global counter by adding the iteration number to it for future calculation of average iteration of each block. Besides, reducer will also update the residual counter for the future calculation of average residual.

5.3 Outer Iteration Control

The while-loop in main function controls mapreduce iteration. When a job has finished, two important numbers will be calculated. One is the average inner iteration number over all blocks, the other one is the average residual of the whole graph, which is used to determine whether to continue mapreduce or not.

6. Gauss-Seidel Blocked PageRank

The only thing we implemented differently from Jacobi is when the PageRank value of a node has been updated, the change of PageRage value for its outgoing edges with the destination node inside this block will be addressed in the list of incoming PageRank values of the destination node.

Comparison between Jacobi and Gauss Seidel:
Gauss Seidel does in-memory computation and is more dynamic in compare with Jacobi.
Also from the EMR result in section 8, the average error does not vary too much, but average number of iterations per block reduced a lot compared to Jacobi.

7. Random Blocked PageRank

To modify the original program to compute random blocked PageRank.
When changed the blockIDofNode() method to be the following:

```
public static long blockIDofNode(long n) {
  return n % 68;
}
```

With the method above, it simply takes the node id as parameter and mod by 68. In this way we can achieve Random Blocked PageRank.

The performance of this program is much slower than the original Blocked PageRank program.

8. Test Note and Result

8.1 How we run EMR

This project runs and is tested on Amazon AWS Elastic MapReduce. To run this project. A jar file needs to be generated from source Java code. Then a EMR cluster need to be initialized with a S3 bucket as the log destination. We test this project with 3 EC2 instances(1 master and 2 slaves). Also, store jar file and input text file into S3.

After EMR cluster is properly initialized, we need to create a step inside of EMR console, pointing jar file location into S3. Also, specify the main class, input text file location and the output location respectively as the three arguments. The arguments are as follow:

```
SimplePageRank
s3://edu-cornell-cs-cs5300s16-ys684-project2/elasticmapreduce/input/inputFile.txt
s3://edu-cornell-cs-cs5300s16-ys684-project2/elasticmapreduce/output

BlockedPageRank
s3://edu-cornell-cs-cs5300s16-ys684-project2/elasticmapreduce/input/inputFile.txt
s3://edu-cornell-cs-cs5300s16-ys684-project2/elasticmapreduce/output

GaussBlockedPageRank
s3://edu-cornell-cs-cs5300s16-ys684-project2/elasticmapreduce/input/inputFile.txt
s3://edu-cornell-cs-cs5300s16-ys684-project2/elasticmapreduce/output

RandomBlockedPageRank
s3://edu-cornell-cs-cs5300s16-ys684-project2/elasticmapreduce/input/inputFile.txt
s3://edu-cornell-cs-cs5300s16-ys684-project2/elasticmapreduce/output
```

Then wait until the step terminates. And we can check the output stdout for the output information. Also detailed stdout information from reducer and mapper can be found at logs folder we specified at the beginning.

8.2 Result of EMR

8.2.1 Simple PageRank

```
SimplePageRank starts running.
Iteration 1 average error 2.430536168877603
Iteration 2 average error 0.3333787764692147
Iteration 3 average error 0.1992263829663033
Iteration 4 average error 0.09728527633057514
Iteration 5 average error 0.06547845278227749
```

8.2.2 Jacobi Blocked PageRank

```
Iteration 1 average error 2.9618181598879207
Iteration 1 average number of iterations per block is
17.86764705882353
Iteration 2 average error 0.03935993170176437
Iteration 2 average number of iterations per block is 7.25
Iteration 3 average error 0.02502320242838171
Iteration 3 average number of iterations per block is
5.926470588235294
Iteration 4 average error 0.010348161201348452
Iteration 4 average number of iterations per block is
4.044117647058823
Iteration 5 average error 0.004170419713089036
Iteration 5 average number of iterations per block is
2.6029411764705883
Iteration 6 average error 0.0010304401441851641
Iteration 6 average number of iterations per block is
1.411764705882353
Iteration 7 average error 7.721608802883703E-4
Iteration 7 average number of iterations per block is
1.2647058823529411
CONVERGED at: 7 iteration
```

8.2.3 Gauss-Seidel Blocked PageRank

```
Iteration 1 average error 2.962526469944398
Iteration 1 average number of iterations per block is
9.852941176470589
Iteration 2 average error 0.04013589524685142
Iteration 2 average number of iterations per block is 5.25
Iteration 3 average error 0.02626268128949404
Iteration 3 average number of iterations per block is
4.588235294117647
Iteration 4 average error 0.011414310961283073
Iteration 4 average number of iterations per block is
3.323529411764706
Iteration 5 average error 0.005180914291551742
Iteration 5 average number of iterations per block is
2.3970588235294117
Iteration 6 average error 0.0019491775024444347
Iteration 6 average number of iterations per block is
1.6764705882352942
Iteration 7 average error 9.358094362476833E-4
Iteration 7 average number of iterations per block is
1.3676470588235294
CONVERGED at: 7 iteration
```

8.2.4 Random Blocked PageRank

```
    Iteration 1 average error 2.4313234251273297
    Iteration 1 average number of iterations per block is 3.0
    Iteration 2 average error 0.3328230382499307
    Iteration 2 average number of iterations per block is
2.735294117647059
    Iteration 3 average error 0.19839586532988923
    Iteration 3 average number of iterations per block is
2.014705882352941
    Iteration 4 average error 0.09662954993213957
    Iteration 4 average number of iterations per block is 2.0
    Iteration 5 average error 0.0646195223501598
    Iteration 5 average number of iterations per block is 2.0
```

The PageRank values of two nodes with smallest node id in Jacobi and Gauss Blocked PageRank program are stored in a separated file called NodePRValues.txt in each blocked pagerank program folder under solution/result.